```
/*

   write_pot(0, 0);   // 7555car Threshold,
  write_pot(1, 0);  // 7555car Discharge,
  write_pot(2, 0);  // 7555car Volume,
  write_pot(3, 0);  // 7555mod vol, effectively modulation index
  write_pot(4, 0);   // 7555mod Threshold
  write_pot(5, 0);   // 7555mod Discharge

when threshold and discharge are moved together on one chip (carrier or modulator)
they control frequency,  when moved independently, they control pitch but with changing
puslewidth or duty cicle.  This changes the timbre of the output.

*/

//Tuning presets.
// scale 0-127 by this much for each pot, volume is scaled differently than other paramete
int data_scale[]= {2, 2, 1.25, 1.25, 2, 2};

// you might need to play with this number to get your controller numbers lined up with th
int controllerOffSet = 0;// this number is specfic to the novation midi controller



int DATAOUT = 11;
int   DATAIN   =   12;//MISO - not used, but part of builtin SPI
int  SPICLOCK   =  13;//sck
int  SLAVESELECT  =  10;//ss
int incomingByte = 0;
int note = 0;
int vel = 0;
int controller = 0;
int value = 0;


byte resistance=0;
int index;
int outdex;



void setup()
{
  //  Set MIDI baud rate:
  Serial.begin(31250);
      // debugger, or serial control baud
  // Serial.begin(9600);

  byte                      i                                          =
  byte clr = 0;
```

```
  //setup pins to work with SPI
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 (fastest)
                                                SPCR                                =
  clr=SPSR;
  clr=SPDR;
  delay(10);

  // flash an LED and flash each digital pot address (for debugging) and to silence the ci

  //Set all pots to minimum/maximum resistance and back again
  for (i=0; i < 6; i++)
  {
    digitalWrite(2, HIGH);
     delay(100);
    write_pot(i,0);
      digitalWrite(2, LOW);
     delay(100);
  }
  delay(1000);  //wait a second


   for (i=0; i < 6; i++)
  {
     digitalWrite(2, HIGH);
     delay(100);
    write_pot(i, 255);
      digitalWrite(2, LOW);
     delay(100);
    // 255 = full resistance...
  }

}




// main program
void loop()
{


  if (Serial.available()   >   0)   {
     incomingByte =Serial.read();
```

```
    // see if there is a specific message format, if a specific MIDI status byte is detec
    //treat the next two incoming bytes as data bytes
   //
      if  (incomingByte   ==   176){// status message for midi cc channel 1
        delay(1);
        // record the next byte as the controller number
         controller =Serial.read()   -   controllerOffSet;
        delay(1);
        // record the third byte as the controller data or value
          value =Serial.read();



      }
      // if the controller number is within the right range,
      // use the controller number to set the address(1 of the 6 resistors) and
      // use the value of that controller to set the resistance of said address

      if (controller < 6 && controller >= 0){

       // some of the parameters may need scaling,
      write_pot( controller, ( (value * data_scale[controller])));

      // turn on an LED when data is sent (for user feedback/debugging)
      digitalWrite(2, HIGH);
      delay(1);
          }// end if controller
    }// end if serial




    delay(1);
    digitalWrite(2, LOW);
}// end main loop

//-------------------------------------------------------------custom function calls--------



// function to send data to a particular address
byte write_pot(int  address,int value)
{
  digitalWrite(SLAVESELECT,LOW);
  //2 byte opcode
  spi_transfer(address);
  spi_transfer(value);
  digitalWrite(SLAVESELECT,HIGH); //release chip, signal end transfer
}
```

```
//function for transfering data over SPI
char  spi_transfer(volatilechar data)
{
           SPDR      =      data;// Start the transmission
  while (!  (SPSR  &  (1  <<  SPIF)))// Wait for the end of the transmission
  {
  };
  return                    SPDR;// return the received byte
}
```